Loops
Sept. 29, Ch 3.3-3.4

[Comic panels showing derivatives of functions and a dialogue]
Announcements

Quiz next week
- Book = okay
- e-book = okay
- notes = okay
- compiler = NO
- internet = NO
- friends = NO

Homework 1 graded (?)
if/else vs loops

if/else statements makes code inside only sometimes run

Loops make code inside run more than once

Both use boolean expressions to determine if the code inside is run
if/else vs loops

General c++ semi-colon rules:

Slap after every command EXCEPT if you are making a block (using { } ) (i.e. after main(), if/else and loops)
A while loop tests a bool expression and will run until that expression is false.

```cpp
while (i < 10) {
    // looped code
    // variable i should change in here
}
```

(See: week 2 whileLoop.cpp)
The **bool** expression is tested when first entering the while loop
And!
When the end of the loop code is reached (the } to close the loop)

```cpp
int i = 0;
while (i < 5) {
    cout << "Looping, i = " << i << "\n";
    i++;
}
cout << "Outside the loop, i = " << i << "\n";
```
while loop

It can be helpful to manually work out what loops are doing and how variables change in each loop iteration.

This will build an insight into how loops work and will be beneficial when working with more complicated loops.
while loop

3 parts to any (good) loop:

- Test variable initialized
  
  ```
  i=0;
  ```

- boolean expression
  
  ```
  while (i < 10)
  ```

- Test variable updated inside loop
  
  ```
  i++;
  ```
A do-while loop is similar to a normal while loop, except the bool expression is only tested at the end of the loop (not at the start)

```cpp
cout << "How many times do you want to run the loop?\n"; cin >> i; // what happens if i is less than 1?
do {
    cout << "Looping, i = " << i << "\n";
    i--;
} while (i > 0); // Note semicolon!
cout << "Outside the loop, i = " << i << "\n";
```

(See: see week 2 doWhile.cpp)
Q: Why would I ever want a do-while loop?

A: When the first time the variable is set is inside the loop. You can initialize the variable correctly and use a normal while loop, but this makes the logic harder.
do-while loop

99 bottles of beer on the wall, 99 bottles of beer!
Take one down, pass it around, 98 bottles of beer on the wall!

98 bottles of beer on the wall, 98 bottles of beer!
Take one down, pass it around, 97 bottles of beer on the wall!

97 bottles of beer on the wall, 97 bottles of beer!
Take one down, pass it around, 96 bottles of beer on the wall!
...

Write a program to output the above song
(see 99beer.cpp)
while vs do-while

(see: vendingV2.cpp)
A for loop is a compacted version of the while loop (the 3 important parts are together)

for loops are used normally when iterating over a sequence of numbers (i.e. 1, 2, 3, 4)

\[\text{for (int } i=0; i < 3; i++)\]

Initialization  boolean expression  Update

(See: forLoop.cpp)
What are the outputs of the following codes?

1. 
```cpp
for(int i=0; i < 10; i*2) {
    cout << i << endl;
}
```

2. 
```cpp
for(int i=0; i < 10; i*=2) {
    cout << i << endl;
    i++;
}
```

3. 
```cpp
for(int i=0; i < 10; i++) {
    for(int j = i; j < 10; j++) {
        cout << j << endl;
    }
    cout << endl;
}
```
```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
Nested for loop

Like nested if statements, we can also make nested loops (which can cause headaches)

It might help to think of each loop as an added dimensions:

1 loop = 1 dimension (line/ruler)
2 loops = 2 dimensions (plane/square/area)
3 loops = 3 dimensions (volume/cube)
...
(See: nestedLoop.cpp)
There are two commands that help control loops:

**continue** tells the loop to start over again

**break** stops the loop
continue command can be issued to start at the next iteration of a loop

```cpp
for (i = 0; i < 10; i++)
{
    // code will run everytime
    if (doSkip)
    {
        continue;
    }
    // code will not run
    // if doSkip is true
}
(See: continue.cpp)
```
C-C-COMBO BREAKER
**break**

**break** will exit the current loop

```cpp
for (i = 0; i < 10; i++)
{
    // code
    if (doSkip)
    {
        break;
    }

    // outside loop code
}
```

(See: break.cpp and vendingV2.cpp)
Infinite loops

(See: countingSheep.cpp)
Loops to sum

Loops allow you to decide how many times a piece of code should run on the fly (i.e. at run time, not compile time)

You can either directly prompt the user how many times or make a special value to “exit” on

(See: sumLoop.cpp)
Debugging

When your program is not working, it is often helpful to add cout commands to find out what is going on.

Normally displaying the value of your variables will help you solve the issue.